Hiding Circuit Components Using Boundary Blurring Techniques

James D. Parham, J. Todd McDonald,
Yong C. Kim and Michael R. Grimaila
Department of Electrical and Computer Engineering
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433-7765
Email: james.parham.2@us.af.mil;{jmcdonal;ykim;mgrimail}@afit.edu

***Abstract*—Protecting circuitry from reverse engineering is extremely important for critical technologies. When systems designed for security become compromised, millions of dollars and countless labor hours may be required for redesigning new protection circuitry. Similarly, an organization using reverse engineering techniques for reproducing systems can do so with significantly lower costs.**

Component identification is an essential step to reverse engineering. Techniques which increase the necessary time for discovering circuit components, and in turn delay or even defeat component identification, increase the level of circuit protection against reverse engineering and other adversarial attacks. In this paper, we discuss our Java based component identification tool implementation. We also introduce two component hiding algorithms and show they effectively defeat component identification.

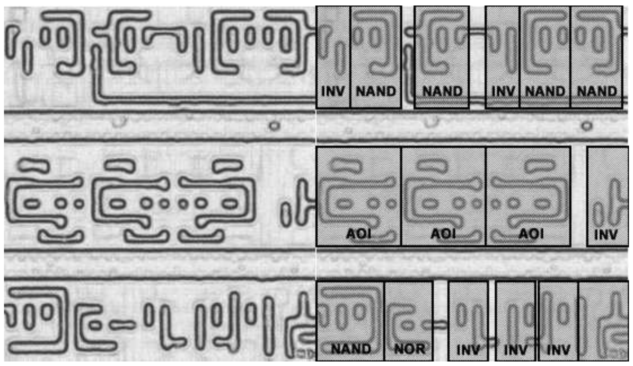

Fig. 1. Exposed transistors of the Mifare RFID are shown on the left and results after gate identification are shown on the right [4].

I. Introduction

Protecting circuitry from reverse engineering is extremely important for many electronic systems. Without implementing some form of protection, an adversary may easily gain access to sensitive information contained within the circuit. For example, the U.S. National Security Agency has authorized the Advanced Encryption Standard (AES) for encrypting classified information on Unmanned Aerial Systems (UAS). Without providing protection to AES circuitry, adversaries could possibly obtain secret keys enabling interception of the data link to substitute their own video feeds or take control of a UAS [1].

Billions of dollars are spent each year developing technologies. The U.S. Department of Defense Fiscal Year 2009 budget request for procurement research and development was in excess of $183 billion for modernizing and meeting future threats [2]. Adversaries gain access to technologies faster and more cheaply when they reverse engineer system circuitry.

Component identification is one method used by reverse engineers for discovering key circuit elements [3]. There are several steps involved for the reverse engineer using this method. First, the circuit of interest must be physically obtained. Next, by exposing the circuit's transistors, the logic gates are identified and then schematics created. From the schematic, circuit architecture and components are identified. Nohl et al. used this technique when revealing the cryptographic cypher of the Mifare Classic Radio Frequency Identification (RFID) tag. They first exposed circuit transistors enabling identification of gate structures and the connections between them. Figure 1 shows a small section of exposed transistors and the results of gate identification. Then they identified the area of the circuit chip containing cryptographic components. Finally, they discovered the cryptographic keys making it possible to access the Dutch transit system where the use of these cards were intended. When they testified before the Dutch government detailing their accomplishments, approximately $2 billion had been invested in the ticketing system [4].

Organizations exist with the sole purpose of discovering the technology inside integrated circuitry with expertise in chip delayering and materials characterization [5]. The real challenge is preventing clustering of such transistors and gates enabling successful component identification and preventing a circuit's intent to be reverse engineered by adversaries or competitors. Delaying or preventing the identification of a circuit's intent is the primary goal of our component hiding techniques.

Prior to our research, a component identification tool was not available for the general public. With the only subcircuit enumeration algorithm known to us, we implemented an identification tool to assist our effort of evaluating our component hiding methods. After implementing the component identification tool, we explored techniques for hiding circuit components. This paper discusses our component identifica-

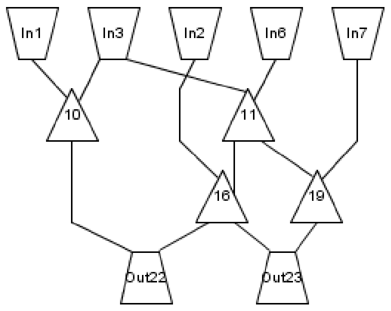

Fig. 2. An five input two output example circuit graph. Triangles represent NAND gates and the remaining shapes represent inputs or outputs.

tion tool and the algorithms developed for effectively hiding components. We also discuss the impact of our protection methods on circuits realized using FPGA technology.

II. Component Identification Tool

We perform component identification in Boolean logic circuits by supplying a circuit under investigation to our Java based identification tool and efficiently enumerating candidate subcircuits for comparison against a known component library. The number of inputs and outputs for the candidate component, or I/O space, is the first comparison for equivalence. We compare candidates and library components with matching I/O space using truth table analysis. All matching components are saved for later reference to the original circuit.

A. Circuit Description Files

We use both BENCH file format and IEEE Std 1076-1993 compliant structural VHDL format for describing Boolean circuits. The BENCH file is a netlist describing circuit gates and the connections between them. Our protection tool imports a circuit by reading this descriptive file. An excerpt of the ISCAS-85 c880 benchmark BENCH file is shown in [3]. Our circuit protection tool generates the synthesizable structural VHDL to allow synthesis and evaluation of circuit performance [6]. In Section II-F we discuss the circuits we used for testing our component identification tool.

B. Modeling Boolean Circuits

We model Boolean circuits using Directed Acyclic Graphs (DAG) where each vertex represents a logic gate and each edge represents the connection between them. Directed graphs are used since flow occurs only in one direction from the output of a gate to the input of one or more gates. Shaped vertices allow visual identification of gate types as well as inputs and outputs. Figure 2 shows an example circuit graph for a five input two output circuit.

C. Component Library

The component library consists of components of interest and is subdivided into directories containing circuits with a specific I/O space. A half adder is a two input two output circuit and is saved in the directory labeled “2-2” while a full adder is a three input two output circuit and is saved in the directory labeled “3-2”.

We developed additional custom components for testing our component identification tool. Each component has an input and output size ranging from two to six. We created the circuits by choosing an appropriate I/O space and randomly generating each output function. Then logic analysis tools synthesize a minimized circuit based upon the randomly generated truth table. The number of gates contained in each component varies; the smallest containing five gates and the largest containing 139.

D. Candidate Subcircuit Enumeration

A fully connected graph has $n!$ subgraphs where n is the number of vertices. A fully connected graph is highly unlikely, but serves as an upper bound for the number of possible sub circuits [7]. This shows that even small circuit graphs contain an intractable number of subgraphs. [8] details a candidate subcircuit enumeration algorithm with runtime $O(n^3)$. Because no source code was available, we implemented our interpretation of their algorithm. The candidate enumeration is rule based and provides focused unique enumeration of components. Each of the candidate components are further classified as fully contained subcircuits meaning each circuit gate has either all or none of its successors and predecessors contained in the candidate graph.

E. Equivalence Checking

When we identify candidates, we perform a check for library modules with matching I/O space. When a match exists, the identification tool performs truth table analysis comparing each truth table column for equivalence. Input and output order effects this analysis, so we compare all input and output orderings. The tool permutes the input and output order until a positive truth table match occurs or all combinations are analyzed. This results in a runtime of $O(n!m!)$ where n is the number of inputs and m is the number of outputs. The runtime limits us to small component I/O space and is why we chose our custom component I/O space.

F. Test Circuits

We used a variety of circuits to test our component identification tool. However, we discuss two significant circuits in this paper. First, is the ISCAS-85 benchmark circuit c6288. This circuit is a 16-bit multiplier composed of 224 full adders and 16 half address and contains 2448 gates. Figure 3 shows a circuit high level diagram. In this benchmark the full adders are realized using NOR gates only resulting in nine intermediate gates. The half adders are realized with NOR gates and inverters and also contains nine intermediate gates. Our tool considers inputs as a component gate so the full adder is a 12 gate component and the half adder is an 11 gate component. We consider the 16-bit multiplier more difficult for hiding components because of its repeated adder structures. If a reverse engineer identifies one adder they most likely can identify all adder components. The ability to hide adders

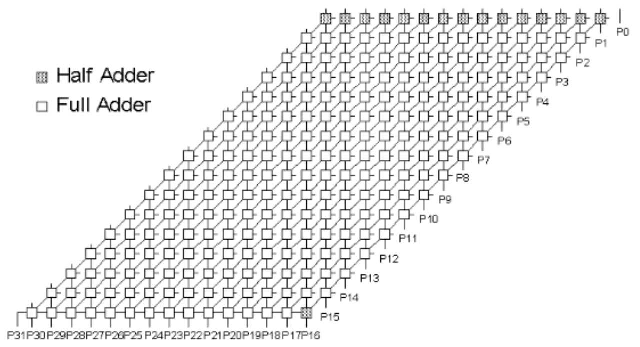

Fig. 3. High level diagram of 16-bit multiplier [9].

in a multiplier makes it possible to hide similar logic within larger circuits with fewer repeated structure where hiding is less challenging.

The second circuit we consider is one of our own custom circuits. Circuit 33-15-555 is a 33 input 15 output circuit composed of 12 components from our custom library and contains 555 gates. The largest component is a six input four output circuit containing 145 gates. We used this circuit for verifying our identification tool can identify components with a relatively large gate size.

Our identification tool identifies all components in both test circuits. This shows the use of computer tools make component identification possible for reverse engineers. We now focus our attention to ways of countering such identification methods.

III. Boundary Blurring Algorithms

The identification tool makes each component input and output (component boundaries) identifiable. Developing a technique which no longer allows a clear separation between components prevents component identification. Any modification to the circuit must maintain the same overall circuit function. Therefore, any changes require recovery within the circuit or additional external circuitry is necessary for recovery of original circuit functions. The next sections discuss types of component boundaries and the two techniques we developed for defeating component identification.

A. Component Boundaries

Components share boundaries with the circuit, with other components or both creating nine different boundary cases. Figure 4 shows a circuit P containing nine components. Each of these components covers one of nine cases. We consider components sharing input and output boundary with the circuit only, case I, as the most difficult for hiding. The 33-15-555 test circuit contains all nine component boundaries.

B. Multilevel Boundary Blurring

Obscuring the boundary between component requires connections between them which does not alter the circuit output. We accomplish this using our Multilevel Blurring technique. This technique uses each identified boundary gate, referred to as a *replacement gate*, as a point for applying the blur. We then determine *recovery gates* by looking at gates three levels closer to the circuit output. If we can not recover three levels closer to the output we reduce this by one until level one, two and three are checked or we determine a suitable recovery level. If no suitable level exists blurring is not applied to the selected replacement gate.

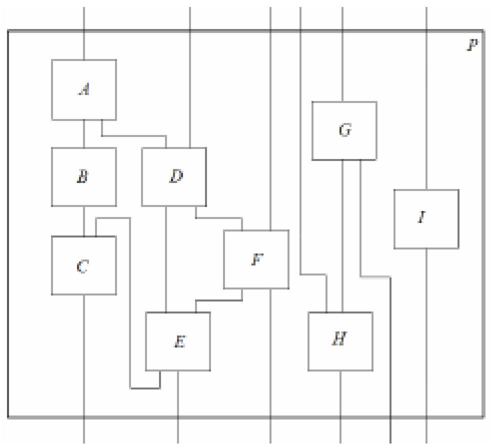

Fig. 4. High level diagram of circuit 33-15-555.

Once we select replacement and recovery gates, a working subcircuit is created using replacement gates as inputs and recovery gates as outputs. We add additional inputs to the subcircuit so all circuit gate successors are part of the subcircuit. Now we record the output signature for each recovery gate. Next we change the replacement gate type to a randomly selected type chosen from the set $\{$AND,NAND,OR,NOR,XOR,XNOR$\}$. This modification causes a change in the signals between the replacement gate and recovery gate. We again record the output signature of the subcircuit. From these two signatures we determine which terms of the modified subcircuit are necessary for recovering the original recovery gate signatures. We use a Quine-McCluskey algorithm for determining a minimized sum of products function which recovers the original signature. Using this function we add necessary gates and connections and for recovering the original signature. The result is two components with connections between them which obscures the component boundary. Table I provides an example of signature before (F_0) and after (F_1) changing a replacement gate's type. The original circuit function is $(A+B)C$ and is modified to ABC. From F_0 you can see terms three, five and seven are the recovery terms which gives a reduced recovery function of $A'BCF_1+ACF_1$.

Multilevel blurring does not make connections between component case I and other circuit components allowing our identification tool to identify case I components in circuits with applied multilevel blurring. This creates a need for a technique which prevents case I identification.

C. Don't Care Boundary Blurring

Don't Care Blurring is a technique which solves the case I component problem. Because circuit output functions are

TABLE I

SIGNALS OF A THREE INPUT CIRCUIT WITH APPLIED MULTILEVEL BLURRING. F_0 IS THE ORIGINAL SIGNAL AND F_1 IS THE MODIFIED SIGNAL.

A	B	C	F_0	F_1
0	0	0	0	0
0	0	0	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	0
1	1	0	0	0
1	1	1	1	1

TABLE II

SIGNALS OF A TWO INPUT CIRCUIT WITH DON'T CARE BLURRING APPLIED. F_0 IS THE ORIGINAL SIGNAL AND F_1 IS THE SIGNAL AFTER THE ADDITION OF THE ADDITIONAL CIRCUIT GATE.

A	B	C	F_0	F_1
0	0	0	0	0
0	0	1	0	0
0	1	0	1	0
0	1	1	1	1
1	0	0	1	0
1	0	1	1	1
1	1	0	1	0
1	1	1	1	1

preserved, the technique is similar to multilevel blurring. However, this technique adds additional inputs to a component which take on a don't care condition. We, as a circuit designer, don't care what the input signal is to the additional input because the subcircuit output is not effected by it.

Each identified boundary gate is a replacement gate for applying the blur. The gate type of a newly introduced gate is referred to as the replacement type. After these selections, we create a subcircuit similar to a multilevel blur. At first, the subcircuit contains only the replacement gate and its inputs. The signature of the gate is recorded. Next, we add a new gate directly to the output of the replacement gate which is randomly chosen from the set $\{$AND,NAND,OR,NOR,XOR,XNOR$\}$. This adds an input to the subcircuit and we now record the new signature. As before, we determine which terms require recovery and perform a Quine-McCluskey reduction enabling the recovery of the original subcircuit signature. The final step is making a random connection between the new input and any other gate output occurring closer to the circuit input.

Table II shows the truth table for the function $F_0 = A + B$. This is a two input circuit containing a single OR gate. When we add a gate with replacement type AND to its output, we create a new function $F_1 = (A + B)C$. From F_0 you can see we must recover terms two through seven. Performing Quine-McCluskey reduction on these terms produces a minimized sum of products function $BC'F_1' + BCF_1 + AC'F_1' + ACF_1$. In this equation the variable C reduces out indicating the input C does not effect the output signature.

IV. Hiding Effectiveness

We measure hiding effectiveness by performing identified boundary blurring using both multilevel and don't care blurring techniques. After the application of blurring, component identification is performed again on the new circuit. The workstation used during our research contains two 2.8Ghz Dual Core Xeons with 4GB of DDR2 memory running Windows XP Pro. The CPU time required to search components in our unprotected circuit is 8736.051 seconds and the protected circuit requires 36357.232 seconds to search. The component size search range for both circuit is a range of 11 to 50 gates. Our goal is zero components identified.

A. Results of Applied Blurring

Multilevel blurring has no effect on the 16-bit multiplier when applied to identified boundary gates. No changes occur due to circuit topology and our blurring implementation. When we take the approach of applying multilevel blurring to gates with maximum fan out 100% hiding is achieved. Applying multilevel blurring to boundaries of the 33-15-555 circuit results in 91.6% hiding. When we apply multilevel blurring to our custom circuit, the tool identifies the case I component.

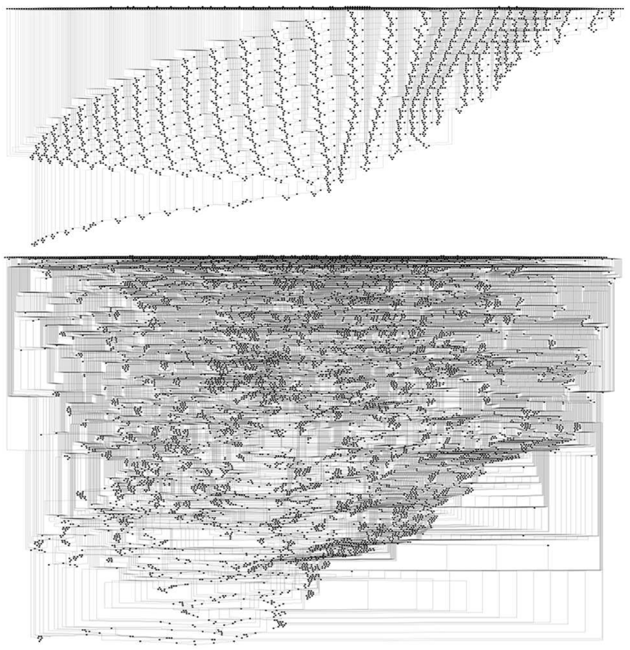

Fig. 5. Structure of 16-bit multiplier before and after boundary blurring. The smaller original circuit is shown above the modified circuit.

Applying don't care blurring to identified boundaries in the 16-bit multiplier results in 100% component hiding. With this blurring method, we do not have to use the maximum fan out approach to achieve 100% hiding. The blurring process increases the circuit gate size by 188% to a total of 7052 gates. Figure 5 shows a before and after view of the multiplier's circuit structure. The upper unprotected circuit clearly shows traceable repeated connections of full adders and half adders, where the protected lower circuit is fully randomized with no detectable full adder and half adder components. 100%

TABLE III
SUMMARY OF COMPONENTS IDENTIFIED USING EACH BLURRING TECHNIQUE.

Circuit	Multilevel	Don't Care
16-bit Multiplier	0	0
33-15-555	1	0

component hiding also occurs in our custom circuit when using the don't care technique. In this circuit variant, all components including the case I are no longer identifiable by the identification tool. Table III summarizes the number of component identified after application of each blurring technique.

B. FPGA Design Considerations

Table IV shows the circuit performance and implementation requirements of the 16-bit multiplier on Xilinx Virtex II Pro FPGA (XC2VP30) which has 30,816 4-bit look up tables (LUTs) for implementing gates or specific components [10]. The original unprotected 16-bit multiplier occupies 1.3% of the available area and operates at 25Mhz. The obfuscated circuit requires 3.6% of area, and operates at 13Mhz due to the increase in gate quantity and levels. These results are provided as a reference to expected overhead if proposed protection methods are applied without optimization. However, it is essential that the protected circuit's primary purpose is in providing a measure of component protection. The proposed method can be applied to only critical portions of a circuit to minimize the overall performance and implementation overhead for the entire circuit.

TABLE IV
PERFORMANCE AND IMPLEMENTATION REQUIREMENTS FOR 16-BIT MULTIPLIER.

	Unprotected Circuit	Protected Circuit
Gates	2448	7052
Number of Slices	174	490
Number of LUTs	348	980
% Area Used	1.27%	3.58%
Operating Frequency	25MHz	13MHz
Clock Period	40ns	77ns
Number of Levels	123	238

V. CONCLUSION

Component identification is an essential element of circuit reverse engineering. We have shown component identification in logic circuits modeled as DAGs is possible using computer tools. Defeating component identification is also possible using our blurring techniques. Multilevel blurring is not as effective in hiding component when the circuit being protected contains components whose boundaries are shared with the circuit boundary. In this situation don't care blurring has the greatest effect. Our experiments applied only one blurring technique to a circuit. Therefore, additional work applying both these methods to a circuit or randomly applying these methods may improve hiding effectiveness.

ACKNOWLEDGMENT

This work was supported by a research grant from the Air Force Office of Scientific Research. The authors would like to thank Dr. Travis Doom at Wright State University for sharing references and discussing his work on component identification with us.

DISCLAIMER

The views expressed in this article are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

REFERENCES

[1] *Unmanned Systems*, vol. 28, Feb 2010.

[2] Office of the Under Secretary of Defense (Comptroller), "Fy 2009 budget request," World Wide Web, 2008. [Online]. Available: http://comptroller.defense.gov/defbudget/fy2009/2009_Budget_Rollout_Release.pdf

[3] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the iscas-85 benchmarks: a case study in reverse engineering," *IEEE Design and Test of Computers*, vol. 16, no. 3, pp. 72 – 80, 1999, carry look ahead;Error correcting circuits;Register transfer;. [Online]. Available: http://dx.doi.org/10.1109/54.785838

[4] K. Nohl, D. Evans, S. Starbug, and H. Plötz, "Reverse-engineering a cryptographic rfid tag," in *SS'08: Proceedings of the 17th conference on Security symposium*. Berkeley, CA, USA: USENIX Association, 2008, pp. 185–193.

[5] Chipworks, "Chipworks inside technology," World Wide Web, 2010. [Online]. Available: http://www.chipworks.com/what_we_do.aspx

[6] IEEE, "Ieee std 1076-1993 ieee standard vhdl language reference manual -description," World Wide Web. [Online]. Available: http://standards.ieee.org/reading/ieee/std_public/description/dasc/1076-1993_desc.html

[7] J. L. White, "Candidate subcircuit enumeration for module identification in digital circuits," Ph.D. dissertation, Department of Computer Science and Engineering, Michigan State University, 2000.

[8] J. L. White, A. S. Wojcik, M.-J. Chung, and T. E. Doom, "Candidate subcircuits for functional module identification in logic circuits," Chicago, IL, USA, 2000, pp. 34 – 38, functional module identification;.

[9] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Iscas-85 c6288 16x16 multiplier," World Wide Web, available at http://www.eecs.umich.edu/ jhayes/iscas/c6288.html.

[10] XILINX, "Virtex-ii pro platform fpgas: Complete data sheet," World Wide Web. [Online]. Available: http://www.digchip.com/datasheets/parts/datasheet/534/XC2VP30-5FF896I-pdf.php